UNITED STATES PATENT AND TRADEMARK OFFICE

UNITED STATES DEPARTMENT OF COMMERCE
United States Patent and Trademark Office
Address: COMMISSIONER FOR PATENTS
P.O. Box 1450
Alexandria, Virginia 22313-1450
www.uspto.gov

APPLICATION NO.	FILING DATE	FIRST NAMED INVENTOR	ATTORNEY DOCKET NO.	CONFIRMATION NO.
09/771,761	01/29/2001	Jeff A. Zimniewicz	MS160268.1	8645

27195 7590 05/17/2007

AMIN. TUROCY & CALVIN, LLP
24TH FLOOR, NATIONAL CITY CENTER
1900 EAST NINTH STREET
CLEVELAND, OH 44114

EXAMINER
YIGDALL, MICHAEL J

ART UNIT	PAPER NUMBER
2192	

MAIL DATE	DELIVERY MODE
05/17/2007	PAPER

Please find below and/or attached an Office communication concerning this application or proceeding.

The time period for reply, if any, is set in the attached communication.

Office Action Summary	Application No.	Applicant(s)	
	09/771,761	ZIMNIEWICZ ET AL.	
	Examiner	Art Unit	
	Michael J. Yigdall	2192	

-- The MAILING DATE of this communication appears on the cover sheet with the correspondence address --

Period for Reply

A SHORTENED STATUTORY PERIOD FOR REPLY IS SET TO EXPIRE 3 MONTH(S) OR THIRTY (30) DAYS, WHICHEVER IS LONGER, FROM THE MAILING DATE OF THIS COMMUNICATION.

- Extensions of time may be available under the provisions of 37 CFR 1.136(a). In no event, however, may a reply be timely filed after SIX (6) MONTHS from the mailing date of this communication.
- If NO period for reply is specified above, the maximum statutory period will apply and will expire SIX (6) MONTHS from the mailing date of this communication.
- Failure to reply within the set or extended period for reply will, by statute, cause the application to become ABANDONED (35 U.S.C. § 133). Any reply received by the Office later than three months after the mailing date of this communication, even if timely filed, may reduce any earned patent term adjustment. See 37 CFR 1.704(b).

Status

1)☒ Responsive to communication(s) filed on *08 March 2007*.

2a)☒ This action is **FINAL**. 2b)☐ This action is non-final.

3)☐ Since this application is in condition for allowance except for formal matters, prosecution as to the merits is closed in accordance with the practice under *Ex parte Quayle*, 1935 C.D. 11, 453 O.G. 213.

Disposition of Claims

4)☒ Claim(s) *1-4,6-18,20-26 and 28-31* is/are pending in the application.

4a) Of the above claim(s) _____ is/are withdrawn from consideration.

5)☐ Claim(s) _____ is/are allowed.

6)☒ Claim(s) *1-4,6-18,20-26 and 28-31* is/are rejected.

7)☐ Claim(s) _____ is/are objected to.

8)☐ Claim(s) _____ are subject to restriction and/or election requirement.

Application Papers

9)☐ The specification is objected to by the Examiner.

10)☐ The drawing(s) filed on _____ is/are: a)☐ accepted or b)☐ objected to by the Examiner.

Applicant may not request that any objection to the drawing(s) be held in abeyance. See 37 CFR 1.85(a).

Replacement drawing sheet(s) including the correction is required if the drawing(s) is objected to. See 37 CFR 1.121(d).

11)☐ The oath or declaration is objected to by the Examiner. Note the attached Office Action or form PTO-152.

Priority under 35 U.S.C. § 119

12)☐ Acknowledgment is made of a claim for foreign priority under 35 U.S.C. § 119(a)-(d) or (f).

a)☐ All b)☐ Some * c)☐ None of:

1.☐ Certified copies of the priority documents have been received.

2.☐ Certified copies of the priority documents have been received in Application No. _____.

3.☐ Copies of the certified copies of the priority documents have been received in this National Stage application from the International Bureau (PCT Rule 17.2(a)).

* See the attached detailed Office action for a list of the certified copies not received.

Attachment(s)

1) ☐ Notice of References Cited (PTO-892)
2) ☐ Notice of Draftsperson's Patent Drawing Review (PTO-948)
3) ☐ Information Disclosure Statement(s) (PTO/SB/08) Paper No(s)/Mail Date _____.
4) ☐ Interview Summary (PTO-413) Paper No(s)/Mail Date. _____ .
5) ☐ Notice of Informal Patent Application
6) ☐ Other: _____.

DETAILED ACTION

1. This Office action is responsive to Applicant's submission filed on March 8, 2007. Claims 1-4, 6-18, 20-26, 28-31 are now pending.

Response to Amendment

2. The examiner notes that claim 7 is identified as "Previously presented" where it should be labeled --Currently Amended--.

Response to Arguments

3. Applicant's arguments have been fully considered but they are not persuasive.

At the outset, one cannot show nonobviousness by attacking references individually where the rejections are based on combinations of references. Furthermore, the test for obviousness is not that the claimed invention must be expressly suggested in any one or all of the references. Rather, the test is what the combined teachings of the references would have suggested to those of ordinary skill in the art. See *In re Keller*, 642 F.2d 413, 208 USPQ 871 (CCPA 1981), and *In re Merck & Co.*, 800 F.2d 1091, 231 USPQ 375 (Fed. Cir. 1986).

Applicant acknowledges that Kawamata discloses an installation order, but contends that Kawamata fails to disclose or suggest either a valid order that is referentially consistent, or an installer for installing or removing components based on the valid order (remarks, page 10).

However, the installation order of Kawamata is in fact a "valid order." It is certainly not an *invalid* order. Kawamata teaches an update sequence management unit 190 that forms an installation sequence table (see, for example, column 9, lines 1-3). "The installation sequence

table 1100 has an installation order 1105 and a name 1110 of software to be installed" (column 9, lines 4-6). Thus, the update sequence management unit 190 forms or generates the installation order. The installation order is generated based on a distribution software list 700 (see, for example, column 8, lines 29-67). Specifically, the update sequence management unit 190 processes the distribution software list 700 to determine a valid installation order (see, for example, column 7, line 58 to column 8, line 25). FIG. 11 illustrates the "contents of the installation sequence table 1100 for the distribution software list 700" (column 9, lines 6-7). The installation order is "referentially consistent" at least because it provides a precise order in which each component is to be installed. For example, "NAVIGATOR UP-DATA VER. 3.0" is "3RD" in the installation order, and is to be installed immediately before "NAVIGATOR UP-DATA VER. 4.0," which is "4TH" in the installation order (FIG. 11). Accordingly, Kawamata teaches a valid order that is referentially consistent.

As set forth in the Office action, it is the Curtis reference that teaches an installer for installing or removing components based on the valid order. Nonetheless, Kawamata discloses, "Next, at Step 630 the software update unit 160 installs the software received by the process from Step 600 to Step 620 in the navigation unit 195, in accordance with the installation order written in the installation sequence table" (column 10, lines 1-4). Clearly, Kawamata also teaches an installer for installing components based on the valid order.

Applicant contends that Noble fails to disclose or suggest a plurality of components that include a shared component, let alone a shared component that subsumes properties of previously installed shared components (remarks, page 10).

However, as set forth in the Office action, it is the Curtis reference that teaches a shared component. Curtis further teaches manipulating at least one property associated with the shared component, but as Applicant notes, does not expressly disclose that the shared component subsumes properties associated with previously installed shared components. Nonetheless, Noble teaches installing components and subsuming properties of previously installed components. In Noble, the "properties" comprise customizations. The new components automatically "subsume" customizations of the previously installed components to save time and reduce costs (see, for example, column 1, lines 45-52 and 58-59). Therefore, Noble suggests to those of ordinary skill in the art implementing Curtis such that the shared component automatically subsumes properties of previously installed shared components.

In response to Applicant's opinion that "one skilled in the art would not arrive at the proposed combination unless guided by a hindsight reading of the subject disclosure" (remarks, page 10), the examiner notes that the reasoning set forth in the Office action takes into account only the teachings of the references themselves and knowledge that was within the level of ordinary skill at the time the claimed invention was made.

Applicant contends that Kruger does not disclose or suggest dependent components associated with a shared component (remarks, page 11).

However, as set forth in the Office action, it is the Curtis reference that teaches dependent components associated with a shared component. Still, the DLL files of Kruger are examples of such shared components. To those of ordinary skill in the art, "the reference count for the DLL file" (column 9, lines 17-18) indicates the number of references to the DLL file. In other words, the reference count indicates the number of dependent components that are associated with the

DLL file. Kruger teaches incrementing and decrementing the reference count based on the installation and removal of components, respectively (see, for example, column 9, lines 14-21). Notwithstanding Applicant's characterization of the reference (remarks, page 11), these operations are not exclusive to font files.

In response to Applicant's arguments directed to amended claims 23, 24 and 31 (remarks, pages 12-13), the examiner notes that as above, the Kruger reference teaches or suggests "incrementing a reference count value indicative of a number of dependent components associated with the at least one shared component." The examiner further notes that claim 31, as amended, does not recite a reference count, let alone incrementing a reference count.

Claim Rejections - 35 USC § 103

4. The following is a quotation of 35 U.S.C. 103(a) which forms the basis for all obviousness rejections set forth in this Office action:

> (a) A patent may not be obtained though the invention is not identically disclosed or described as set forth in section 102 of this title, if the differences between the subject matter sought to be patented and the prior art are such that the subject matter as a whole would have been obvious at the time the invention was made to a person having ordinary skill in the art to which said subject matter pertains. Patentability shall not be negatived by the manner in which the invention was made.

5. Claims 1, 6-13, 16, 18, 20-22, 25, 26 and 28 are rejected under 35 U.S.C. 103(a) as being unpatentable over U.S. Patent No. 6,442,754 to Curtis (art of record, "Curtis") in view of U.S. Patent No. 6,820,259 to Kawamata et al. (art of record, "Kawamata"), in view of U.S. Patent No. 5,845,128 to Noble et al. (art of record, "Noble"), and in view of U.S. Patent No. 6,367,075 to Kruger et al. (art of record, "Kruger").

With respect to claim 1 (currently amended), Curtis discloses a system that facilitates installation and/or removal of components (see, for example, the title and abstract) that includes at least one shared component (see, for example, column 4, lines 33-37, which shows components that are depended upon by more than one program, i.e. shared components), comprising:

(a) a validation engine that provides a valid order (see, for example, column 11, lines 11-20, which shows a function for checking dependencies, i.e. a validation engine, and column 12, lines 22-32, which shows generating a list of dependent components and providing a valid installation order).

Curtis does not expressly disclose that the order is referentially consistent relative to each of the components.

However, in an analogous art, Kawamata discloses a validation engine (see, for example, software update sequence management unit 190 in FIG. 1) that provides a valid order of components that is referentially consistent relative to each of the components (see, for example, installation order 1105 in FIG. 11 and column 9, lines 1-16). The valid order enables a plurality of components to be installed without any dependency problems among them (see, for example, column 14, lines 43-48).

It would have been obvious to one of ordinary skill in the art at the time the invention was made to modify the Curtis system such that the valid order is referentially consistent relative to each of the components, such as taught by Kawamata, so as to avoid dependency problems among the components.

Curtis further discloses:

(b) an installer that controls at least one of an install and removal operation of the components based on the valid order, the installer manipulates at least one property associated with the at least one shared component to reflect dependency for the at least one shared component relative to one or more other components that depend on the shared component according to the installation or removal thereof (see, for example, column 12, lines 32-50, which shows an installer for installing the components based on the valid order, and see, for example, FIG. 5 and column 13, lines 7-10, which shows a data structure having properties that reflect dependency of one or more components that depend on a shared component, and column 13, lines 28-29, which shows manipulating the data structure when a component is installed).

Curtis does not expressly disclose that the at least one shared component automatically subsumes one or more property associated with previously installed shared components.

However, in an analogous art, Noble discloses installing new components to replace previously installed components (see, for example, column 6, lines 45-52). The new components automatically subsume one or more properties associated with the previously installed components (see, for example, column 6, lines 53-62), so as to automatically apply any previously made customizations (see, for example, column 1, lines 45-52 and 58-59).

It would have been obvious to one of ordinary skill in the art at the time the invention was made to modify the Curtis system such that the at least one shared component automatically subsumes one or more property associated with previously installed shared components, such as taught by Noble, so as to automatically apply any previously made customizations.

Although Curtis shows storing dependency information in order to indicate which components depend on a shared component (see, for example, column 13, lines 1-6), Curtis does

not expressly disclose that the at least one property further comprises a reference count having a value indicative of a number of dependent components associated with the at least one shared component.

However, Kruger discloses the limitation above in terms of an installer that uses a reference count for shared library files, to ensure that files depended upon by other programs are not affected (see, for example, column 9, lines 14-21).

It would have been obvious to one of ordinary skill in the art at the time the invention was made to modify the Curtis system with the reference count feature taught by Kruger, for the purpose of ensuring that shared components used by other programs are not affected, inherently reducing the number of potential version conflicts.

With respect to claim 6 (currently amended), the rejection of claim 1 is incorporated, and Kruger further suggests that the installer increases the value of the reference count for each installation of the at least one shared component (see, for example, column 9, lines 14-21, which shows incrementing the reference count when a file is added or installed).

With respect to claim 7 (currently amended), the rejection of claim 1 is incorporated, and Kruger further suggests that the installer decreases the value of the reference count in response to removal of a dependent component that depends on the at least one shared component (see, for example, column 9, lines 14-21, which shows decrementing the reference count when a file is deleted or removed).

With respect to claim 8 (previously presented), the rejection of claim 1 is incorporated, and Curtis further discloses that the at least one property further comprises configuration data

that indicates an operating relationship of the at least one shared component and each installed dependent component associated with the at least one shared component (see, for example, FIG. 5 and column 13, lines 7-27, which shows a data structure having properties that indicate the relationship between a component and its dependencies).

With respect to claim 9 (previously presented), the rejection of claim 1 is incorporated, and Curtis further discloses that the installer controls installation of the at least one shared component, such that a single set of files for the at least one shared component is copied as part of the installation for use by associated dependent components (see, for example, column 9, lines 47-64, which shows determining whether dependencies are already installed and installing a set of files for a shared component).

With respect to claim 10 (previously presented), the rejection of claim 9 is incorporated, and Curtis further discloses that the at least one shared component has associated metadata that identifies the at least one shared component as a shared component (see, for example, FIG. 3 and column 9, lines 10-25, which shows a dependency object comprising metadata that identifies whether a component is a shared component).

With respect to claim 11 (previously presented), the rejection of claim 10 is incorporated, and Curtis further discloses that the at least one shared component requires at least one dependent component to perform a substantially useful function (see, for example, column 9, lines 25-31, which shows that dependent components must be installed in order for another component to perform all intended functions).

With respect to claim 12 (previously presented), the rejection of claim 11 is incorporated, and Curtis further discloses that a runtime dependency exists between an installed dependent component and the shared component on which the dependent component depends (see, for example, column 9, lines 39-43, which shows dependencies needed by a component in order to operate, i.e. dependencies needed at runtime).

With respect to claim 13 (currently amended), Curtis discloses a system that effectuates installation of components (see, for example, the title and abstract) including at least one shared component (see, for example, column 4, lines 33-37, which shows components that are depended upon by more than one program, i.e. shared components), comprising:

(a) a setup manager that controls installation of the components (see, for example, column 5, lines 56-60, which shows an installer script, i.e. a setup manager);

(b) dependency manager that provides a valid installation order based on metadata associated with at least some of the components (see, for example, column 11, lines 11-20, which shows a function for checking dependencies, i.e. a dependency manager, using dependency objects, and FIG. 3 and column 9, lines 10-25, which shows that the dependency objects comprise metadata, and see, for example, column 12, lines 22-32, which shows generating a list of dependent components and providing a valid installation order).

Curtis does not expressly disclose that the valid installation order is generated to ensure that relative dependencies between the components are resolved prior to installation.

However, in an analogous art, Kawamata discloses a dependency manager (see, for example, software update sequence management unit 190 in FIG. 1) that provides a valid installation order of components (see, for example, installation order 1105 in FIG. 11 and

column 9, lines 1-16). The valid installation order is generated to ensure that relative dependencies among the components are resolved prior to installation, so as to install the components without any dependency problems (see, for example, column 14, lines 43-48).

It would have been obvious to one of ordinary skill in the art at the time the invention was made to modify the Curtis system such that the valid installation order is generated to ensure that relative dependencies between the components are resolved prior to installation, such as taught by Kawamata, so as to avoid dependency problems among the components.

Curtis further discloses that the setup manager causes the components to be installed according to the valid installation order, a separate shared installation of the at least one shared component being implemented for each dependent component that depends on the at least one shared component (see, for example, column 12, lines 32-50, which shows installing each component based on the valid installation order).

Although Curtis shows storing dependency information in order to indicate which components depend on a shared component (see, for example, column 13, lines 1-6, and note that the information is written during installation), Curtis does not expressly disclose that the separate shared installation comprises incrementing a reference count value indicative of a number of dependent components associated with the at least one shared component.

However, Kruger discloses the limitation above in terms of incrementing a reference count when a file is added or installed, to ensure that files depended upon by other programs are not affected (see, for example, column 9, lines 14-21).

It would have been obvious to one of ordinary skill in the art at the time the invention was made to modify the Curtis system with the reference count feature taught by Kruger, for the

purpose of ensuring that shared components used by other programs are not affected, inherently reducing the number of potential version conflicts.

Curtis does not expressly disclose that the separate shared installation comprises configuring properties of the at least one shared component, which preserves properties associated with previously installed shared components.

However, in an analogous art, Noble discloses installing new components to replace previously installed components (see, for example, column 6, lines 45-52). The new components preserve properties associated with the previously installed components (see, for example, column 6, lines 53-62), so as to automatically apply any previously made customizations (see, for example, column 1, lines 45-52 and 58-59).

It would have been obvious to one of ordinary skill in the art at the time the invention was made to modify the Curtis system such that the at least one shared component preserves properties associated with previously installed shared components, such as taught by Noble, so as to automatically apply any previously made customizations.

With respect to claim 16 (previously presented), the rejection of claim 13 is incorporated, and the features recited in the claim are analogous to those of claim 10 (see the explanation for claim 10 above).

With respect to claim 18 (previously presented), the rejection of claim 13 is incorporated, and Curtis further discloses at least one property associated with an installed instance of the at least one shared component which reflects dependency for the at least one shared component

(see, for example, FIG. 5 and column 13, lines 7-10, which shows a data structure having properties that reflect dependency).

With respect to claim 20 (currently amended), the rejection of claim 13 is incorporated, and the features recited in the claim are analogous to those of claims 6 and 7 (see the explanations for claims 6 and 7 above).

With respect to claim 21 (previously presented), the rejection of claim 13 is incorporated, and the features recited in the claim are analogous to those of claim 8 (see the explanation for claim 8 above).

With respect to claim 22 (previously presented), the rejection of claim 13 is incorporated, and the features recited in the claim are analogous to those of claim 9 (see the explanation for claim 9 above).

With respect to claim 25 (currently amended), Curtis discloses a system that effectuates installation and/or removal of components (see, for example, the title and abstract) including at least one shared component (see, for example, column 4, lines 33-37, which shows components that are depended upon by more than one program, i.e. shared components), comprising:

(a) means for providing a valid order for the components (see, for example, column 12, lines 22-32, which shows generating a list of dependent components and providing a valid installation order).

Curtis does not expressly disclose that the valid order is referentially consistent relative to each of the components.

However, in an analogous art, Kawamata discloses a means for providing a valid order (see, for example, software update sequence management unit 190 in FIG. 1) that provides a valid order of components that is referentially consistent relative to each of the components (see, for example, installation order 1105 in FIG. 11 and column 9, lines 1-16). The valid order enables a plurality of components to be installed without any dependency problems among them (see, for example, column 14, lines 43-48).

It would have been obvious to one of ordinary skill in the art at the time the invention was made to modify the Curtis system such that the valid order is referentially consistent relative to each of the components, such as taught by Kawamata, so as to avoid dependency problems among the components.

Curtis further discloses:

(b) means for controlling installation of the components based on the valid order (see, for example, column 12, lines 32-50, which shows an installer for installing the components based on the valid order); and

(c) means for manipulating at least one property associated with the at least one shared component to reflect dependency for the at least one shared component relative to one or more dependent components that depend on the shared component based on at least one installation of the shared component and removal of a dependent component (see, for example, FIG. 5 and column 13, lines 7-10, which shows a data structure having properties that reflect dependency of one or more components that depend on a shared component, and column 13, lines 28-29, which shows manipulating the data structure when a component is installed).

Curtis does not expressly disclose that prior to replacement of a currently installed shared component with the at least one shared component, the at least one shared component automatically acquires previously affixed properties associated with the currently installed shared component.

However, in an analogous art, Noble discloses installing new components to replace currently installed components (see, for example, column 6, lines 45-52). The new components automatically acquire one or more properties associated with the currently installed components (see, for example, column 6, lines 53-62), so as to automatically apply any previously made customizations (see, for example, column 1, lines 45-52 and 58-59).

It would have been obvious to one of ordinary skill in the art at the time the invention was made to modify the Curtis system such that prior to replacement of a currently installed shared component with the at least one shared component, the at least one shared component automatically acquires previously affixed properties associated with the currently installed shared component, such as taught by Noble, so as to automatically apply any previously made customizations.

Although Curtis shows storing dependency information in order to indicate which components depend on a shared component (see, for example, column 13, lines 1-6), Curtis does not expressly disclose that the at least one property further comprises a reference count value that indicates of a number of dependent components.

However, Kruger discloses the limitation above in terms of an installer that uses a reference count for shared library files, to ensure that files depended upon by other programs are not affected (see, for example, column 9, lines 14-21).

It would have been obvious to one of ordinary skill in the art at the time the invention was made to modify the Curtis system with the reference count feature taught by Kruger, for the purpose of ensuring that shared components used by other programs are not affected, inherently reducing the number of potential version conflicts.

With respect to claim 26 (currently amended), Curtis discloses a method for installing and/or removing components (see, for example, the title and abstract) including at least one shared component (see, for example, column 4, lines 33-37, which shows components that are depended upon by more than one program, i.e. shared components), the method comprising:

(a) providing a valid order (see, for example, column 12, lines 22-32, which shows generating a list of dependent components and providing a valid installation order).

Curtis does not expressly disclose that the valid order is consistent relative to each of the components.

However, in an analogous art, Kawamata discloses providing a valid order of components that is consistent relative to each of the components (see, for example, installation order 1105 in FIG. 11 and column 9, lines 1-16), which enables a plurality of components to be installed without any dependency problems among them (see, for example, column 14, lines 43-48).

It would have been obvious to one of ordinary skill in the art at the time the invention was made to modify the Curtis system such that the valid order is consistent relative to each of the components, such as taught by Kawamata, so as to avoid dependency problems among the components.

Curtis further discloses:

(b) installing each of the plurality of components based on the valid order (see, for example, column 12, lines 32-50, which shows installing the components based on the valid order).

Although Curtis shows storing dependency information in order to indicate which components depend on a shared component (see, for example, column 13, lines 1-6, and note that the information is written during installation), Curtis does not expressly disclose:

(c) incrementing a reference count value indicative of each of the plurality of components associated with the at least one shared component.

However, Kruger discloses the limitation above in terms of incrementing a reference count when a file is added or installed, to ensure that files depended upon by other programs are not affected (see, for example, column 9, lines 14-21).

It would have been obvious to one of ordinary skill in the art at the time the invention was made to modify the Curtis system with the reference count feature taught by Kruger, for the purpose of ensuring that shared components used by other programs are not affected, inherently reducing the number of potential version conflicts.

Curtis further discloses:

(d) modifying at least one property associated with the at least one shared component to reflect dependency characteristics of the at least one shared component relative dependent components that utilize the at least one shared component (see, for example, FIG. 5 and column 13, lines 7-10, which shows a data structure having properties that reflect dependency, and column 13, lines 28-29, which shows manipulating the data structure when a component is installed).

Curtis does not expressly disclose that the at least one shared component acquires existing properties of shared components that the at least one shared component supercedes on installation.

However, in an analogous art, Noble discloses installing new components to replace existing components (see, for example, column 6, lines 45-52). The new components acquire one or more properties of the existing components (see, for example, column 6, lines 53-62), so as to automatically apply any previously made customizations (see, for example, column 1, lines 45-52 and 58-59).

It would have been obvious to one of ordinary skill in the art at the time the invention was made to modify the Curtis system such that the at least one shared component acquires existing properties of shared components that the at least one shared component supercedes on installation, such as taught by Noble, so as to automatically apply any previously made customizations.

With respect to claim 28 (original), the rejection of claim 26 is incorporated, and the features recited in the claim are analogous to those of claims 6 and 7 (see the explanations for claims 6 and 7 above).

6. Claim 2-4, 14, 15, 17, 29 and 30 are rejected under 35 U.S.C. 103(a) as being unpatentable over Curtis in view of Kawamata in view of Noble and in view of Kruger, as applied to claims 1, 13 and 26 above, respectively, and further in view of U.S. Patent No. 5,721,824 to Taylor (art of record, "Taylor").

With respect to claim 2 (previously presented), the rejection of claim 1 is incorporated, and although Curtis shows that shared, or dependent, components are identified for installation prior to non-shared components (see, for example, column 12, lines 27-32), Curtis does not expressly disclose that the valid order identifies shared components for installation subsequent to non-shared components.

However, Taylor discloses the limitation above in terms of an action list, i.e. a valid order, that identifies shared, or dependent, components for installation subsequent to a non-shared package (see, for example, column 5, lines 25-29). Note that Taylor also discloses an implementation wherein dependent components are identified for installation before non-shared components, as in the Curtis system (see, for example, column 7, lines 49-53).

It would have been obvious to one of ordinary skill in the art at the time the invention was made to modify the Curtis system with the feature of identifying shared components for installation subsequent to non-shared components, as taught by Taylor, for the purpose of supporting an installation sequence that conforms to the constraints of the target system (see, for example, Taylor, column 2, lines 1-3), in order to increase the compatibility of the installation routine with different platforms.

With respect to claim 3 (previously presented), the rejection of claim 2 is incorporated, and Curtis further discloses that the installer initiates a method to install each of the components based on the valid order during a part of the installation (see, for example, column 12, lines 32-50, which shows an installer for installing each of the components based on the valid order, and column 12, lines 59-62, which shows that the installer is operative to initiate the installation).

Curtis does not expressly disclose that the at least one shared component is installed and configured for a selected dependent component during the first part of installation.

However, Taylor further discloses the limitation above in terms of installing components based on the action list, i.e. the valid order, during a first part of the installation (see, for example, column 2, lines 7-11, which shows that the flow of operations is layered, i.e. has multiple parts, and column 2, lines 12-26, which shows installing packages during a first part of the installation).

It would have been obvious to one of ordinary skill in the art at the time the invention was made to modify the Curtis system with the feature of installing components during a first part of the installation, as taught by Taylor, for the purpose of installing multiple software packages with a single load on the system, in order to improve the perceived performance (see, for example, Taylor, column 3, lines 47-51).

With respect to claim 4 (previously presented), the rejection of claim 3 is incorporated. Curtis does not disclose that the method employs a second part of the installation to install the at least one shared component for each dependent component other than the selected dependent component during the second part of the installation.

However, Taylor further discloses the limitation above in terms of installing packages or components that are depended upon by other dependent packages during a second part of the installation (see, for example, column 2, lines 7-11, which shows that the flow of operations is layered, i.e. has multiple parts, and column 2, lines 53-62, which shows installing packages during a second part of the installation).

It would have been obvious to one of ordinary skill in the art at the time the invention was made to modify the Curtis system with the feature of installing components during a second part of the installation, as taught by Taylor, for the purpose of installing multiple software packages with a single load on the system, in order to improve the perceived performance (see, for example, Taylor, column 3, lines 47-51).

With respect to claim 14 (previously presented), the rejection of claim 13 is incorporated, and although Curtis shows a dependency manager for generating a valid installation order (see, for example, column 11, lines 11-20, which shows a function for checking dependencies, i.e. a dependency manager, and column 12, lines 22-32, which shows generating a list of dependent components and providing a valid installation order), Curtis does not expressly disclose that the dependency manager validates a received installation order, which, upon validation of the received installation order, becomes the valid installation order.

However, Taylor further discloses the limitation above in terms of validating a dependency list, i.e. a received installation order, and using it as the valid installation order (see, for example, column 2, lines 28-40, which shows translating the dependency list into an action list, i.e. a valid installation order).

It would have been obvious to one of ordinary skill in the art at the time the invention was made to modify the Curtis system with the feature of validating an installation order, as taught by Taylor, for the purpose of installing multiple software packages with a single load on the system, in order to improve the perceived performance (see, for example, Taylor, column 3, lines 47-51).

With respect to claim 15 (previously presented), the rejection of claim 14 is incorporated, and although Curtis shows a dependency manager for generating a valid installation order (see, for example, column 11, lines 11-20, which shows a function for checking dependencies, i.e. a dependency manager, and column 12, lines 22-32, which shows generating a list of dependent components and providing a valid installation order), Curtis does not expressly disclose that when the received installation order is improper, the dependency manager creates the valid installation order.

However, Taylor further discloses the limitation above in terms of validating a dependency list, i.e. a received installation order, and creating a valid installation order (see, for example, column 2, lines 12-26, which shows generating an action list, i.e. a valid installation order).

It would have been obvious to one of ordinary skill in the art at the time the invention was made to modify the Curtis system with the feature of validating an installation order, as taught by Taylor, for the purpose of installing multiple software packages with a single load on the system, in order to improve the perceived performance (see, for example, Taylor, column 3, lines 47-51).

With respect to claim 17 (previously presented), the rejection of claim 13 is incorporated, and although Curtis shows an installer or setup manager for installing each of the components based on the valid installation order (see, for example, column 12, lines 32-50, and see, for example, column 12, lines 59-62, which shows that the installer is operative to initiate the installation), Curtis does not expressly disclose that the setup manager initiates a method to install each of the components according to the valid installation order during a first part of the

installation, the at least one shared component being installed for a first dependent component during the first part of installation, the method installs the at least one shared component for each other dependent component during a second part of the installation.

However, Taylor further discloses the limitation above in terms of installing components based on the action list, i.e. the valid installation order, during a first part of the installation, and installing components that are depended upon by other dependent packages during a second part of the installation (see, for example, column 2, lines 7-11, which shows that the flow of operations is layered, i.e. has multiple parts, and see, for example, column 2, lines 12-26, which shows installing packages during a first part of the installation, and column 2, lines 53-62, which shows installing packages during a second part of the installation).

It would have been obvious to one of ordinary skill in the art at the time the invention was made to modify the Curtis system with the feature of first and second installation parts, as taught by Taylor, for the purpose of installing multiple software packages with a single load on the system, in order to improve the perceived performance (see, for example, Taylor, column 3, lines 47-51).

With respect to claim 29 (original), the rejection of claim 26 is incorporated, and the features recited in the claim are analogous to those of claim 3 (see the explanation for claim 3 above).

With respect to claim 30 (original), the rejection of claim 29 is incorporated, and the features recited in the claim are analogous to those of claim 4 (see the explanation for claim 4 above).

7. Claim 23 and 24 are rejected under 35 U.S.C. 103(a) as being unpatentable over Curtis in view of Kawamata in view of Taylor in view of Kruger and in view of Noble.

With respect to claim 23 (currently amended), Curtis discloses a system that facilitates installation and/or removal of components (see, for example, the title and abstract) including at least one shared component (see, for example, column 4, lines 33-37, which shows components that are depended upon by more than one program, i.e. shared components), comprising:

(a) a validation component that provides a valid order based on setup data (see, for example, column 11, lines 11-20, which shows a function for checking dependencies, i.e. a validation component, using dependency objects, and FIG. 3 and column 9, lines 10-25, which shows that the dependency objects comprise setup data, and see, for example, column 12, lines 22-32, which shows generating a list of dependent components and providing a valid installation order).

Curtis does not expressly disclose that the validation component receives a list of components to be installed and/or removed that is organized in an invalid order, and resolves all relative inter-component conflicts prior to installation and/or removal of components.

However, in an analogous art, Kawamata discloses a validation component (see, for example, software update sequence management unit 190 in FIG. 1) that receives a list of components to be installed that is organized in an invalid order (see, for example, software distribution list 700 in FIG. 7) and provides a valid order of components (see, for example, installation order 1105 in FIG. 11 and column 9, lines 1-16), so as to resolve all relative inter-

component conflicts prior to installation and thus enable the components to be installed without any dependency problems among them (see, for example, column 14, lines 43-48).

It would have been obvious to one of ordinary skill in the art at the time the invention was made to modify the Curtis system such that the validation component receives a list of components to be installed and/or removed that is organized in an invalid order, and resolves all relative inter-component conflicts prior to installation and/or removal of components, such as taught by Kawamata, so as to avoid dependency problems among the components.

Although Curtis shows an installer or setup engine for installing each of the components based on the valid installation order (see, for example, column 12, lines 32-50, and column 12, lines 59-62, which shows that the installer is operative to initiate the installation), Curtis does not expressly disclose:

(b) a setup engine that initiates installation of each of the components according to the valid order during a first part of the installation, the shared component being installed for a first dependent component during the first part of installation, the shared component being installed for each other dependent component during a second part of the installation separate from the first part.

However, Taylor discloses the limitation above in terms of installing components based on an action list, i.e. a valid installation order, during a first part of the installation, and installing components that are depended upon by other dependent packages during a second part of the installation (see, for example, column 2, lines 7-11, which shows that the flow of operations is layered, i.e. has multiple parts, and see, for example, column 2, lines 12-26, which shows

installing packages during a first part of the installation, and column 2, lines 53-62, which shows installing packages during a second part of the installation).

It would have been obvious to one of ordinary skill in the art at the time the invention was made to modify the Curtis system with the feature of first and second installation parts, as taught by Taylor, for the purpose of installing multiple software packages with a single load on the system, in order to improve the perceived performance (see, for example, Taylor, column 3, lines 47-51).

Although Curtis shows storing dependency information in order to indicate which components depend on a shared component (see, for example, column 13, lines 1-6, and note that the information is written during installation), Curtis does not expressly disclose the second part of installation comprising incrementing a reference count value indicative of a number of dependent components associated with the at least one shared component.

However, Kruger discloses the limitation above in terms of incrementing a reference count when a file is added or installed, to ensure that files depended upon by other programs are not affected (see, for example, column 9, lines 14-21).

It would have been obvious to one of ordinary skill in the art at the time the invention was made to modify the Curtis system with the reference count feature taught by Kruger, for the purpose of ensuring that shared components used by other programs are not affected, inherently reducing the number of potential version conflicts.

Curtis does not expressly disclose that the second part of the installation comprising configuring properties of the shared component, which automatically retains a property associated with a previously installed shared component.

However, in an analogous art, Noble discloses installing new components to replace previously installed components (see, for example, column 6, lines 45-52). The new components automatically retain one or more properties associated with the previously installed components (see, for example, column 6, lines 53-62), so as to automatically apply any previously made customizations (see, for example, column 1, lines 45-52 and 58-59).

It would have been obvious to one of ordinary skill in the art at the time the invention was made to modify the Curtis system such that the shared component being installed automatically retains a property associated with a previously installed shared component, such as taught by Noble, so as to automatically apply any previously made customizations.

With respect to claim 24 (currently amended), Curtis discloses a system that facilitates installation and/or removal of components (see, for example, the title and abstract) including at least one shared component (see, for example, column 4, lines 33-37, which shows components that are depended upon by more than one program, i.e. shared components), comprising:

(a) a dependency manager that provides a valid order based on setup data (see, for example, column 11, lines 11-20, which shows a function for checking dependencies, i.e. a dependency manager, using dependency objects, and FIG. 3 and column 9, lines 10-25, which shows that the dependency objects comprise setup data, and see, for example, column 12, lines 22-32, which shows generating a list of dependent components and providing a valid installation order).

Curtis does not expressly disclose that the dependency manager receives a list of components to be installed and/or removed that is organized in a random order, and resolves all relative inter-component conflicts prior to installation and/or removal of the components.

However, in an analogous art, Kawamata discloses a dependency manager (see, for example, software update sequence management unit 190 in FIG. 1) that receives a list of components to be installed that is organized in a random order (see, for example, software distribution list 700 in FIG. 7) and provides a valid order of components (see, for example, installation order 1105 in FIG. 11 and column 9, lines 1-16), so as to resolve all relative inter-component conflicts prior to installation and thus enable the components to be installed without any dependency problems among them (see, for example, column 14, lines 43-48).

It would have been obvious to one of ordinary skill in the art at the time the invention was made to modify the Curtis system such that the dependency manager receives a list of components to be installed and/or removed that is organized in a random order, and resolves all relative inter-component conflicts prior to installation and/or removal of the components, such as taught by Kawamata, so as to avoid dependency problems among the components.

Although Curtis shows an installer or setup engine for installing each of the components based on the valid installation order (see, for example, column 12, lines 32-50, and column 12, lines 59-62, which shows that the installer is operative to initiate the installation), Curtis does not expressly disclose:

(b) a setup engine that initiates installation of each of the components according to the valid order during a first part of the installation, the shared component being installed for a first dependent component during the first part of installation, the shared component being installed for each other dependent component during a second part of the installation, which is subsequent to the first part.

However, Taylor discloses the limitation above in terms of installing components based on an action list, i.e. a valid installation order, during a first part of the installation, and installing components that are depended upon by other dependent packages during a second part of the installation (see, for example, column 2, lines 7-11, which shows that the flow of operations is layered, i.e. has multiple parts, and see, for example, column 2, lines 12-26, which shows installing packages during a first part of the installation, and column 2, lines 53-62, which shows installing packages during a second part of the installation).

It would have been obvious to one of ordinary skill in the art at the time the invention was made to modify the Curtis system with the feature of first and second installation parts, as taught by Taylor, for the purpose of installing multiple software packages with a single load on the system, in order to improve the perceived performance (see, for example, Taylor, column 3, lines 47-51).

Although Curtis shows storing dependency information in order to indicate which components depend on a shared component (see, for example, column 13, lines 1-6), Curtis does not expressly disclose that a reference count value is maintained corresponding to a number of dependent components associated with the at least one shared component.

However, Kruger discloses the limitation above in terms of an installer that uses a reference count for shared library files, to ensure that files depended upon by other programs are not affected (see, for example, column 9, lines 14-21).

It would have been obvious to one of ordinary skill in the art at the time the invention was made to modify the Curtis system with the reference count feature taught by Kruger, for the

purpose of ensuring that shared components used by other programs are not affected, inherently reducing the number of potential version conflicts.

Curtis does not expressly disclose that the shared component being installed for the first dependent component acquires properties associated with a shared component previously installed and associated with the first dependent component.

However, in an analogous art, Noble discloses installing new components to replace previously installed components (see, for example, column 6, lines 45-52). The new components acquire one or more properties associated with the previously installed components (see, for example, column 6, lines 53-62), so as to automatically apply any previously made customizations (see, for example, column 1, lines 45-52 and 58-59).

It would have been obvious to one of ordinary skill in the art at the time the invention was made to modify the Curtis system such that the shared component being installed for the first dependent component acquires properties associated with a shared component previously installed and associated with the first dependent component, such as taught by Noble, so as to automatically apply any previously made customizations.

Curtis further discloses:

(c) a setup manager that manipulates at least one property associated with the at least one shared component to reflect dependency characteristics of the at least one shared component as a function of at least one of installation of the shared component and removal of a dependent component that depends on the at least one shared component (see, for example, FIG. 5 and column 13, lines 7-10, which shows a data structure having properties that reflect dependency

characteristics, and column 13, lines 28-29, which shows manipulating the data structure when a component is installed).

8. Claim 31 is rejected under 35 U.S.C. 103(a) as being unpatentable over Curtis in view of Kawamata in view of Taylor and in view of Noble.

With respect to claim 31 (currently amended), Curtis discloses a method to facilitate installing and/or removing components (see, for example, the title and abstract) including at least one shared component (see, for example, column 4, lines 33-37, which shows components that are depended upon by more than one program, i.e. shared components), the method comprising:

(a) providing a valid order (see column 12, lines 22-32, which shows generating a list of dependent components and providing a valid installation order).

Curtis does not expressly disclose that the valid order is referentially consistent relative to each of the installed and/or removed components.

However, in an analogous art, Kawamata discloses providing a valid order of components that is referentially consistent relative to each of the components (see, for example, installation order 1105 in FIG. 11 and column 9, lines 1-16). The valid order enables a plurality of components to be installed without any dependency problems among them (see, for example, column 14, lines 43-48).

It would have been obvious to one of ordinary skill in the art at the time the invention was made to modify the Curtis system such that the valid order is referentially consistent relative to each of the installed and/or removed components, such as taught by Kawamata, so as to avoid dependency problems among the components.

Although Curtis shows an installer for installing each of the components based on the valid order (see, for example, column 12, lines 32-50, and column 12, lines 59-62, which shows that the installer is operative to initiate the installation), Curtis does not expressly disclose:

(b) effecting installation of each of the components during a first part of installation according to the valid order, the shared component being installed for a first dependent component during the first part of the installation;

(c) effecting installation of the shared component for each other dependent component during a second part of the installation separate from the first part.

However, Taylor discloses step (b) above in terms of installing components based on an action list, i.e. a valid order, during a first part of the installation (see, for example, column 2, lines 7-11, which shows that the flow of operations is layered, i.e. has multiple parts, and column 2, lines 12-26, which shows installing packages during a first part of the installation).

Taylor further discloses step (c) above in terms of installing packages or components that are depended upon by other dependent packages during a second part of the installation (see, for example, column 2, lines 7-11, which shows that the flow of operations is layered, i.e. has multiple parts, and column 2, lines 53-62, which shows installing packages during a second part of the installation).

It would have been obvious to one of ordinary skill in the art at the time the invention was made to modify the Curtis system with the feature of first and second installation parts, as taught by Taylor, for the purpose of installing multiple software packages with a single load on the system, in order to improve the perceived performance (see, for example, Taylor, column 3, lines 47-51).

Curtis does not expressly disclose that the shared component acquires one or more extant properties associated with a shared component made obsolete and removed by installation of the shared component.

However, in an analogous art, Noble discloses installing new components to replace obsolete components (see, for example, column 6, lines 45-52). The new components acquire one or more properties associated with the obsolete components (see, for example, column 6, lines 53-62), so as to automatically apply any previously made customizations (see, for example, column 1, lines 45-52 and 58-59).

It would have been obvious to one of ordinary skill in the art at the time the invention was made to modify the Curtis system such that the shared component acquires one or more extant properties associated with a shared component made obsolete and removed by installation of the shared component, such as taught by Noble, so as to automatically apply any previously made customizations.

Curtis further discloses:

(d) manipulating at least one property associated with the at least one shared component to reflect dependency for the at least one shared component relative to at least one dependent component according to the installation or removal thereof (see, for example, FIG. 5 and column 13, lines 7-10, which shows a data structure having properties that reflect dependency of one or more components that depend on a shared component, and column 13, lines 28-29, which shows manipulating the data structure when a component is installed).

Conclusion

9. Applicant's amendment necessitated the new ground(s) of rejection presented in this Office action. Accordingly, **THIS ACTION IS MADE FINAL**. See MPEP § 706.07(a). Applicant is reminded of the extension of time policy as set forth in 37 CFR 1.136(a).

A shortened statutory period for reply to this final action is set to expire THREE MONTHS from the mailing date of this action. In the event a first reply is filed within TWO MONTHS of the mailing date of this final action and the advisory action is not mailed until after the end of the THREE-MONTH shortened statutory period, then the shortened statutory period will expire on the date the advisory action is mailed, and any extension fee pursuant to 37 CFR 1.136(a) will be calculated from the mailing date of the advisory action. In no event, however, will the statutory period for reply expire later than SIX MONTHS from the date of this final action.

10. Any inquiry concerning this communication or earlier communications from the examiner should be directed to Michael J. Yigdall whose telephone number is (571) 272-3707. The examiner can normally be reached on Monday through Friday from 7:30am to 4:00pm.

If attempts to reach the examiner by telephone are unsuccessful, the examiner's supervisor, Tuan Q. Dam can be reached on (571) 272-3695. The fax phone number for the organization where this application or proceeding is assigned is 571-273-8300.

Information regarding the status of an application may be obtained from the Patent Application Information Retrieval (PAIR) system. Status information for published applications may be obtained from either Private PAIR or Public PAIR. Status information for unpublished applications is available through Private PAIR only. For more information about the PAIR system, see http://pair-direct.uspto.gov. Should you have questions on access to the Private PAIR system, contact the Electronic Business Center (EBC) at 866-217-9197 (toll-free). If you would like assistance from a USPTO Customer Service Representative or access to the automated information system, call 800-786-9199 (IN USA OR CANADA) or 571-272-1000.

MY

Michael J. Yigdall
Examiner
Art Unit 2192

mjy

TUAN DAM
SUPERVISORY PATENT EXAMINER